SDE Assignment

Frontend (React + Vite) & Backend (Python + FastAPI)

Use Case: Employee Directory Search System

Your company wants a small internal tool to **search and view employees**.

Each employee has:

- `id`
- `name`
- `email`
- `department`
- `designation`
- `date_of_joining`

The application should allow users to **search employees by name or department** efficiently.

Tech Stack

- **Frontend**: React + Vite
- **Backend**: Python + FastAPI
- **Database**: Preferably **MySQL**

Frontend Requirements (4 Micro Tasks)

Frontend Task 1 – Employee Search (Performance-Oriented)

Create a search bar to find employees by **name or department**.

🔹 Requirements:

- Typing in the search bar should **not cause an API request for every character**
- API requests should be sent in a controlled, optimized manner
- Search results should update smoothly

🔹 Example:

Typing `"Rah"` should not trigger 3 separate API calls immediately

Frontend Task 2 – API State Handling

Handle all API states clearly:

- Loading employees
- Showing results
- No employees found
- Error while fetching data

🔹 Requirements:

- Show loaders/messages where appropriate
- Errors must be visible and user-friendly

Frontend Task 3 – Component Structure & Reusability

Design reusable components such as:

- `SearchBar`
- `EmployeeCard`
- `EmployeeList`

🔹 Requirements:

- Avoid placing API logic directly inside UI-heavy components
- Use props and state correctly

Frontend Task 4 – Environment Configuration

Use environment variables for:

- Backend API base URL

🔹 Requirements:

- No hardcoded API URLs
- Should work across different environments

Backend Requirements (4 Micro Tasks)

Backend Task 1 – Database Design & Connection (Core Task)

Design a database to store employee data.

🔹 Requirements:

- Prefer **MySQL**
- Create a proper `employees` table
- Use a safe and scalable method to connect to the database
- Clearly explain **why this approach is suitable**

Backend Task 2 – Employee Search API

Create an API endpoint such as:

```
GET /employees?search=rahul
```

🔹 Requirements:

- Should search by name or department
- Must be efficient even if employee count grows large
- Avoid fetching unnecessary data

Backend Task 3 – Validation & Error Handling

Add request validation and meaningful errors.

- Examples:
  - Empty search query
  - Invalid query parameters
  - Database connection failure
- Requirements:
  - Use proper HTTP status codes
  - Clear error messages

Backend Task 4 – Clean Architecture

Structure backend code using:

- Routers
- Services
- Database layer / repository

- Requirements:
  - Business logic should not live inside route handlers
  - Easy to extend (e.g., adding employee creation later)

Submission Instructions

1. GitHub repository containing:
   - `frontend/`
   - `backend/`
2. `README.md` explaining:
   - Setup steps

    - Database choice
    - How search performance is optimized (conceptually)
3. `.env.example` (no secrets)

Evaluation Criteria

Skill Area	Importance
Problem-Solving & Thinking	⭐⭐⭐⭐
Code Quality & Structure	⭐⭐⭐⭐
Performance Awareness	⭐⭐⭐
API & DB Design	⭐⭐⭐
Documentation	⭐⭐

Disallowed

- Hardcoded credentials or URLs
- Uncontrolled API calls
- Mixing DB logic inside routes
- Copy-paste code without understanding